Efficient Range Distribution Query for Visualizing Scientific Data

Abon Chaudhuri[1], Tzu-Hsuan Wei[1], Teng-Yok Lee[1], Han-Wei Shen[1], Tom Peterka[2]

[1]The Ohio State University*
[2]Argonne National Laboratory†

ABSTRACT

Visualization applications implicitly run queries on the data to retrieve distributions and statistical measures derivable from distributions. Distribution based data summaries can substitute for the raw data to answer statistical queries of different kinds. However, frequent access to the raw data is no longer practical, if possible at all, for answering large number of queries on large-scale data. Our work addresses the issue by accelerating *range distribution query*, which returns the distribution of an axis-aligned query region. Maintaining the interactivity of such query results is a challenging task because the workload, which affects the response time, of such queries scales up with the data and the query size. In this paper, we present a framework for answering range distribution queries for any arbitrary region in near constant time, regardless of data and query size. We adapt an integral histogram based data structure to bound the workload which is a combination of computation, I/O and communication cost. We propose two novel transformations of this data structure – a decomposition and a similarity-driven indexing – to reduce the huge storage cost associated with it. In addition to studying performance of query, we also demonstrate the benefits that our technique offers to visualization applications which directly or indirectly require distributions.

1 INTRODUCTION

Query-driven techniques have become increasingly popular for exploring and visualizing scientific datasets. Statistical summaries such as mean, standard deviation, entropy and higher order moments computed from distributions of spatial sub-ranges are useful for feature extraction, uncertainty quantification, multi-resolution analysis, and data reduction, just to name a few. Useful local statistics are often derived from distributions commonly represented as histograms. In general, such queries are known as *range distribution query* [14], which returns the distribution of an axis-aligned query region of any arbitrary size. Traditionally, such queries are supported by running a sequential scan through the raw data. However, as the data size grows, frequently accessing large subsets of the raw data increases workload, which leads to slower query response time. This is because both the time and space complexity of such queries are proportional to the number of data points in the query region.

The importance of range distribution query is apparent from its wide use, directly or indirectly, in a number of visualization algorithms [22, 17, 6] (explained in more detail in Section 2). Since the approach of scanning raw data does not scale well with data size, it is important to devise strategies to maintain the interactivity of a distribution query engine. In this paper, we propose to transform the raw data into a data structure which stores and uses pre-computed distributions to answer distribution query for any range in logarithmic time in the worst case. However, we need to address two major challenges to make this approach effective:

- **Query workload:** The workload of the query engine should be low and constant regardless of the query size and data size. The workload refers to a combination of the cost of accessing data in memory, the I/O cost associated with reading files from the disk, if needed, and also the communication cost of exchanging partial query results, if needed.
- **Distribution storage:** The number of pre-computed distributions required to answer range distributions queries is as large as the number of data points. We must minimize the storage cost of the pre-computed distributions so that the query can be run on commodity desktops with moderate amount of storage.

To bound the query workload, we adapt a data structure called integral distribution [19]. It is an extension of the summed area table (SAT) proposed by Crow [5] which, at each location of the data, stores the sum of the attribute values from the origin up to that location. SAT can compute the sum of an attribute in any rectangular region in constant time, regardless of the region size. Similarly, integral distribution can compute the distribution of any query region by accessing pre-computed distributions only at the corner points of the query region. Hence, the workload is bounded by the number of corner points which is same for any query of any size.

The benefit, however, comes at the cost of huge storage cost. The total space required to store an integral histogram is in the order of $O(N*K)$ for N data points and K number of bins per histograms. This is prohibitively large even for data sets of moderate size. We address this storage issue by proposing a set of transformations to stored integral distributions to make them compression-friendly. After the proposed transformations - a decomposition followed by an indexing - are run on the integral distributions, any off-the-shelf compression technique can be employed to achieve much higher space saving. We propose the following transformations for integral distributions: First, we decompose them into a hierarchy of block distributions pertaining to power-of-two length sub-ranges. We propose a novel similarity-based indexing technique to be employed on these block distributions. These transformations lead to significant space saving and much faster query performance.

2 RELATED WORK

Distribution-based Techniques: The use of distributions in data analysis and visualization is ever-growing. Scientific simulations often produce outcomes with uncertainty, making a distribution-based approach the most suitable for post-processing [22]. Sometimes the data itself come in the form of distributions. For example, multi-run simulations generate ensemble data which produce distribution of the outcomes from individual runs. To analyze data of this type, various algorithms that use distribution queries have been developed. Examples include fuzzy

*The email addresses of Abon Chaudhuri, Tzu-Hsuan Wei, and Han-Wei Shen are {chaudhua, weit, hwshen}@cse.ohio-state.edu. The email address of Teng-Yok Lee is Teng-Yok.T.Lee@ieee.org.

†The email address of Tom Peterka is tpeterka@mcs.anl.gov.

isosurface computation from uncertain data [22], and analysis of uncertain vector fields [17]. Even for conventional data, Johnson and Huang have proposed a distribution-based query to identify regions having a certain type of distribution [10]. Similarly, Gosink et al. have shown that distributions of sub-regions of varying sizes are useful for feature extraction and segmentation [6]. Block level distributions have also been used in spatio-temporal data analysis [7]. ProbVis [20] is another distribution query engine. However, all these methods access the raw data to compute and display distributions whereas we aim at answering queries without having to access the raw data. Hence, our method can extend any of these query-driven techniques to large data when raw data access is not recommended.

Distributions of scalar values and their attributes play a critical role in designing transfer function for volume rendering [11, 13] and predicting isosurface statistics [1, 3, 16]. Distributions are also critical in information theory based visualization since local and global distributions are needed to compute a suite of information theoretic metrics [23].

Efficient Storage of Distributions: Exact query processing on large databases became infeasible especially for certain applications such as online analytical processing (OLAP), giving way to fast and approximate query return based on pre-computed distributions [2]. In another approach, histograms have been treated as signals and various basis transforms such as wavelet [15] and SVD [18] have been used to compress them. In visualization community, a recent work has proposed a statistical method of approximating histograms by Gaussian Mixture kernels to reduce the storage overhead [12]. Another recent work has converted local distributions to sparse representations to reduce the overhead of computing probabilistic mipmaps from gigapixel images [8]. Efficient storage and retrieval of histograms is studied in other areas such as computer vision [4] as well.

Similarity-based Methods: The presence of self-similarity in various types of data has been used for feature detection, encoding and other purposes. Shechtman and Irani [21] have utilized self-similarity within an image and across multiple sequences in a video frame to detect visual entities. Jacquin [9], introduced a self-similarity based coding technique, known as fractal coding, for images. In fractal coding, the image to be encoded is partitioned into spatial blocks called *range blocks* (R_k). Another smaller set of partitions, called *domain blocks*, of the same image is created to serve as representative templates. The range block set is mapped to the domain block set in such a way that the range blocks, and hence the data can be reconstructed only from a set of iterative functions.

In this paper, we observe the great extent of similarity present across distributions coming from the same data. and develop a similarity-base matching framework for representing histograms in scientific data sets.

3 Proposed Framework

In this paper, we present a framework to rapidly return the distribution of any arbitrarily large axis-aligned spatial region in a volumetric data. When such queries are answered using the raw data, the response time is a function of the size and shape of the query region, which is manageable for relatively smaller data which fits in memory. However, in most real applications, the data is too big for memory and hence, is partitioned and stored in the disk as multiple blocks. In this case, not only the query region is too big for a sequential scan, it spans across multiple data blocks, requiring numerous files to be loaded to answer it. As a result, the time required to scan the data as well as the I/O cost associated with loading these data blocks add to the query workload. Figure 1 (left) schematically presents the problem. Moreover, if the data blocks are loaded in different processors in a distributed memory system, communication

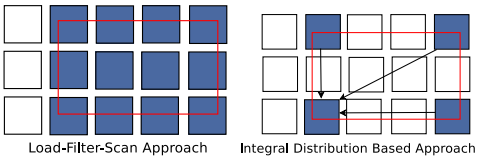


Figure 1. **Left.** Traditional raw data access method of answering range distribution queries where compute time, i/o and communication time are functions of query size. The blocks to be loaded to answer a query (red rectangle) are shaded in blue. **Right.** Proposed integral distribution approach only needs to probe the 4 (8 in 3D) corners of a query region to answer it. Query response time is fixed regardless of query size.

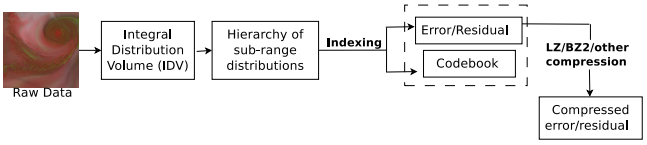


Figure 2. Overview of the proposed framework for transformation and storage of integral distribution volume.

is needed among processors to aggregate the locally computed partial distributions. This communication cost also adds to the workload which is reflected in the response time.

Our framework offers the ability to answer such a range distribution query in near constant cost (in terms of computation time, I/O and/or communication cost), regardless of the query region size. As outlined in Figure 1 (right), our framework can answer any query by probing only the 2^d corner points (8 for 3D) of a d-dimensional query. Hence, the I/O cost is bounded by that of 8 blocks and at most 8 communications are needed.

To give an overview, our framework, outlined in Figure 2, transforms the raw data into a data structure called Integral Distribution Volume (IDV). Since direct computation of IDV is slow and space-consuming for large data, we propose a scheme to compute the integral distribution of any location from a hierarchy of reusable block-level distributions. The high storage cost associated with IDV is reduced by applying a template-based encoding technique on the block-level distributions. Also, we allow the user to trade accuracy for speed by selectively using or discarding block-level distributions while answering query.

Our proposed framework is explained in following stages: representation of integral distribution by power-of-two lengths sub-range distributions, indexing of these sub-ranges followed by compression, and finally, interactive retrieval and reconstruction of range distributions.

3.1 Integral Distribution Volume (IDV)

Given an $N_x \times N_y \times N_z$ 3D field $f : R^3 \rightarrow R$, its integral volume I_f stores at each location the aggregate of the attribute value from origin to that location. Mathematically speaking,

$$I_f = \{I_f(x,y,z) : x \in [1,N_x], y \in [1,N_y], z \in [1,N_z]\}, \text{ where}$$

$$I_f(x,y,z) = \sum_{k=1}^{z} \sum_{j=1}^{y} \sum_{i=1}^{x} f(x,y,z)$$

In other words, the integral volume stores the range prefix sums of f at each data point. As a result, the aggregate of f over any range Q can be obtained by a finite number of addition and subtractions of the prefix sums at the corners of Q [5, 19].

Integral Distribution Volume is an extension of the above idea to histograms. Given an $N_x \times N_y \times N_z$ 3D field, its *integral distribution volume (IDV)* stores at each location the integral histogram, which is the distribution of the sub-field spanning from the origin to that

location. The IDV of a volume under K-bin histogram representation is defined as:

$$IDV_f = \{H(x,y,z) : x \in [1,N_x], y \in [1,N_y], z \in [1,N_z]\}, \text{ where}$$

$H(x,y,z)$ denotes the integral histogram at (x,y,z). The distribution of any 1D range $[A,B]$, denoted by $h([A,B])$, is obtained by using the integral distributions $H(A)$ (same as $h([1,A])$) and $H(B)$ (same as $h([1,B])$): $h([A,B]) = H(B) - H(A)$. In higher dimensions, the query is defined by more than two points (4 points in 2D, 8 in 3D for example). Hence, the distribution of a d-dimensional query range $\{A_i : i = [1,2^d]\}$ is retrieved by the following steps: $H(A_i)$ for each corner point A_i are accessed from IDV; $H(i)$s are added or subtracted to obtain $h(\{A_i : i = [1,2^d]\})$.

3.2 Transformation to Sub-range Distributions

The benefit of near constant time query of IDV comes at the cost of huge storage cost - $O(N_x \times N_y \times N_z \times K)$. Instead of directly applying an off-the-shelf compression to IDV, we propose to first decompose it into a number of sub-ranges and their distributions for two reasons. First, these sub-range distributions can be repeatedly used to reconstruct the integral distribution at any location on the IDV. Second, compressing the sub-range distributions leads to more space-saving than directly compressing the IDV.

Martin and Shen [14] observed that given any 1D point P, the integral distribution $H(P)$ can be computed by combining $\log_2 P$ or less number of sub-range distributions from $[1,P]$, where each sub-range has a power-of-two length. Suppose, $S(P)$ denotes the set of power-of-two sub-ranges for a point P. In this paper, we propose a fast bitwise operation based algorithm for computing $S(P)$ for any P. The steps of the proposed algorithm are:

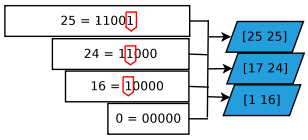


Figure 3. Fast algorithm to decompose a range into power-of-two length blocks.

1. The rightmost non-zero bit of the binary representation of P is set to zero to obtain P' ($P' < P$)
2. $[(P'+1),P]$ is included in $S(P)$ as the next sub-range
3. Steps 1 and 2 are recursively applied on the residual, $(P-P')$
4. The recursion stops when $P' = 0$

Figure 3 demonstrates the algorithm with an example. Starting from $P = 25$ which is 11001 in binary, we follow the above steps to generate the following numbers in sequence: 11000 (24), 10000 (16) and 00000 (0). Hence, the resulting sequence of sub-ranges are $(25,25)$, $(24,17)$ and $(1,16)$.

When P is a higher dimensional point, the 1D blocks for each of its dimensions are first computed. Then, another iteration is performed to combine them into a set of higher dimensional power-of-two length blocks. Hence, $S(P(x,y,z)) = S(x) \times S(y) \times S(z)$. For example, $S(5)$ contains $(1,4)$ and $(5,5)$; $S(4)$ contains $(1,4)$. Hence, the decomposition of the 2D point $(5,4)$ consists of the following 2D sub-ranges: $\{(1,4),(1,4)\}$ and $\{(5,5),(1,4)\}$.

Integral distributions of nearby points share sub-ranges between them. For example, $S(4)$ contains $(1,4)$, which also appears in $S(5)$ which is $\{(1,4),(5,5)\}$ and $S(6)$ which is $\{(1,4),(5,6)\}$. Hence, after running the above algorithm for entire range from 1 to P, we discard the duplicate blocks and retain the union of all minus the duplicates.

Conceptually, the sub-ranges are nodes of a space-partitioning tree. In 1D, they come from a binary tree which partitions the range $[1,P]$. Now, it turns out that storing every alternate sub-range at each level of the binary tree is sufficient to reconstruct $H(P)$ for any arbitrary P. To give an example, Figure 4 shows using blue arrows how a query range $(4,13)$ can be reconstructed using sub-ranges. $H(13)$ is reconstructed from $(1,8),(9,12)$ and $(13,13)$ and H(3) is computed using $(1,2)$ and $(3,3)$. Figure 4 shades the blocks which are sufficient to reconstruct any integral distribution in the range 1 to 16. In 2D or 3D also, the number of sub-ranges to be retained is same as the range itself. In other words, given a dataset of dimension N^3, distributions of N^3 sub-ranges from a space-partitioning tree are to be retained.

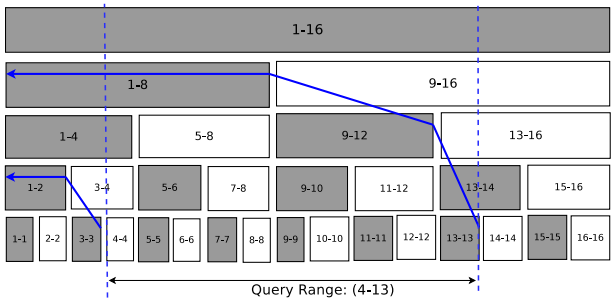


Figure 4. Transformation of a 1D integral distribution. Only the distributions of the ranges colored in gray are sufficient to construct the distribution of any arbitrary range.

The above transformation of IDV creates a set containing same number of sub-range distributions as the IDV does. However, these sub-range distributions are non-integral, and hence can be computed much faster. More importantly, most sub-ranges represent small spatial regions (half of them only contain 1 data point), and are likely to have a very small number of non-zero bins. So they can be stored much more compactly and lend themselves more easily to various compression schemes. When we need to store non-normalized distributions, the range of frequencies of the sub-range distributions is much smaller compared to the original integral distributions. This also leads to better compression.

3.3 Indexing of Sub-Range Distributions

We propose to index the sub-range distributions in such a way that the indexed distributions lead to even more space saving under any standard compression algorithm. In essence, the set of sub-range distributions, denoted by B, is indexed by a much smaller set of template distributions, denoted by T, which represents different types of distributions in the dataset. The proposed algorithm finds a mapping $\Theta : B \rightarrow T$ from each element $h_i(B)$ of B to one or more elements of $h_j(T)$ of T. At the end, B is discarded; only the map and the template set is retained. Since $B >> T$, this reduces the storage cost. In the query phase, the required elements of B are reconstructed using the map and their images in T. Figure 5 presents a schematic overview of the indexing algorithm.

A template based indexing performs well if the elements of B are similar to each other so that many of them are indexed to one template. This reduces the number of elements to retain from T and hence, reduces the total storage cost. In our case, the created sub-range distributions can be seen as a group of sub-ranges of varying sizes (in terms of number of voxels it covers) such as the ones containing 1, 2, 4 or 8 voxels and so on (B_1, B_2 etc. in Figure 5). Having such a grouped

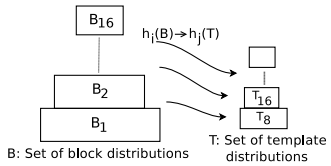


Figure 5. Overview of the indexing algorithm, where sub-range distributions, grouped based on their sizes, find an approximate match from a much smaller set of template distributions.

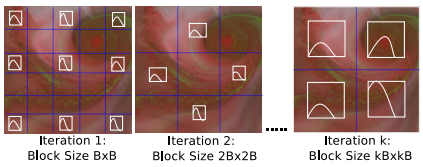


Figure 6. Proposed strategy for template creation. Regions of length B at regular intervals are chosen as templates in the first iteration. Region length is increased and the interval is decreased in subsequent iterations.

structure is beneficial for our next step which is grouping the similar ones. An integral distribution volume in its original form would not contain such groups, since every integral distribution would come from a different-sized sub-range.

3.3.1 Construction of Templates

In our formulation, the distributions to be encoded (set B) come only from power-of-2 length sub-ranges. This is why we create a similarly structured yet much smaller set of template distributions T from power-of-2 length sub-ranges obtained from a downsampled copy of the data. Spatially smoothed downsampled data is used so that the local features and noises do not bias the templates. The templates are created using the following steps:

1. The data is downsampled by one level using a standard spatial smoothing technique. We start with an empty set T.
2. The downsampled data domain is decomposed into non-overlapping partitions of length B where B is a power of two. We have started with 4 as the initial value of B.
3. Every k-th partition along each dimension is selected (where k denotes stride) and its distribution is included in a set T_b
4. T_b is added to T
5. Steps 2 and 3 are repeated with a double length and a half stride, i.e., $b = 2 \times b$ and $k = max\{k/2, 1\}$. This is done until B is nearly half as the full data range.

Figure 6 presents a schematic view of the algorithm. This algorithm populates T only with power-of-two sub-ranges of different sizes. Like B, the template set also contains groups coming from same-sized templates (as shown by T_8, T_16 in Figure 5). This makes it fit for indexing the hierarchy of power-of-two sub-range distributions B. Even though a sub-range distribution is free to index against any template coming from any region in the spatial domain, it is more likely to find a match from the subset coming from sub-ranges of equivalent size in the downsampled data.

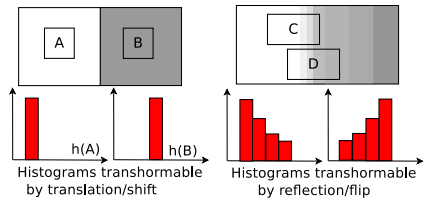
3.3.2 Mapping between Sub-ranges and Templates

Algorithm: The next step is to create the map $\Theta : B \rightarrow T$. For each sub-range histogram h_B, the goal is to find a template which best approximates it. However, since $|T| << |B|$, a h_B may not find a good match in T. Thus, we allow each template to undergo a set of transformations to virtually expand the set of templates available for each h_B. The template which best approximates a sub-range histogram under some transformation becomes the index of that sub-range. Hence, the mapping can be expressed as following:

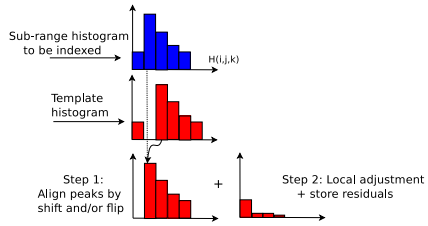
$$\Theta(h_i(B)) = \{j, \tau\}, \text{ if } \Delta(h_i(B), \tau(h_j(T)) \text{is minimum} \forall j$$

where τ represents the transformation of a template and Δ represents the difference between the sub-range histogram and the transformed template.

Transformations: We observed that *circular shift* and *reflection* form a suitable set of transformations for the templates. These two capture all possible shifts of the bins, while preserving the relative order. The intuition behind choosing these transformations is



(a) Intuition behind circular shift and reflection.



(b) Peak matching followed by local alignment

Figure 7. (a) Schematic presentation of chosen transformations for histograms. (b) Fast shift and reflection through peak matching and local adjustment.

twofold. First, circular shift corresponds to change of location in the data space. In a volume data composed of multiple materials, when we move from one uniform zone A, made of one material, to another zone B made of a different material, the data values shift and so does the histograms ($h(A)$ shifts to $h(B)$ as in Figure 7a). Second, data sets also contain regions with gradually changing values. Change of location in such regions corresponds to reflection of histograms. For example, in Figure 7a, the histograms of C and D can be transformed though reflection followed by translation. In case of a 1D histogram with K bins, the shifts are produced by shifting every bin to its right by one step at each iteration. The rightmost bin is circularly brought back to the leftmost bin. Then, if necessary, the histogram is reflected so that for each i, $f(i)$ becomes $f(K-i)$. Then K circular shifts are applied on the reflected histogram. Hence, K or $2 \times K$ transformations can be produced for each template.

Implementation: Given a sub-range, examining all possible transformations for all templates can be prohibitively slow. In practice, we use a reduced search space. First, we first perform a shift to align the modes (bin with highest frequency) of the sub-range and the template histogram. In the next step, we generate a few more transformed configurations of the template by shifting it only by a few steps (we used 3) in both directions about the mode (Figure 7b right). Hence, total number of transformations that a template goes through comes down to $(1+7)$ or $2 \times (1+7)$ if reflection is used.

Under each transformation, the transformed template histogram is compared against the sub-range distribution under consideration. We have used $L2$-norm for this comparison. $L2$-norm is $\Sigma_{i=1}^{K} d_i^2$, where d_i represents the difference between the frequencies at i^{th} bins of two histograms under consideration. After the sub-range has been compared with all templates and all transformations, the $<$*template,transformation*$>$ pair leading to minimum $L2$-norm (Δ) is stored. Hence, transformation $\tau(h)$ involves two pieces of information: the amount or shift n_R, and a boolean flag b_R denoting if reflection is needed. We also store the *residuals* - the bin-wise frequency differences between the block distribution and template chosen for it.

3.3.3 Compression of Indexing Results

The output of the above two steps is a $<$*template,transformation, residual*$>$ triplet for each sub-range distribution. Due to indexing,

each of these three components has become friendly to compression. The template id can take values only within the range 0 to $(|T|-1)$, which is not large. The rotation amount can vary only between 0 and $(K-1)$, if K bins are used. The reflection flag is only 0 or 1. Most importantly, the residuals are likely to be zeros or very small numbers since the it has already been aligned with a transformed template to minimize frequency differences.

We store each of these information in separate files. They can be compressed using any off-the-shelf compression technique. We have experimented with a few lossless compression schemes such as LZ [24] and bzip2. However, our work is not tied up to any particular compression technique, we have focused on increasing the compressibility of the data in general.

3.4 Range Distribution Query

To answer a range distribution query for region Q, we first obtain the integral distribution - $H(Q_i)$ - at each corner point Q_i of the query. To get $H(Q_i)$, we need to access and de-compress the codebook of the block that contains Q_i. We subdivide the range $[1,Q_i]$ into a set of sub-ranges denoted by $S(Q_i)=\{q_1,q_2,...,q_i\}$ using the algorithm described before. Distribution for each q_i is retrieved from the codebook. The decoding phase also requires the set of template distributions T to be loaded. Retrieval of a $h(q_j)$ from the codebook involves the following steps:

1. The codebook is used to retrieve the index to a template, say k and a transformation τ
2. Template histogram $h_k(T)$ is retrieved from template set T
3. The transformation is applied to the template histogram to obtain $\tau(h_k(T))$
4. The stored residuals δ for this mapping is retrieved, decompressed if needed, and applied to obtain $h(q_j)=\tau(h_k(T))+\delta$.

Addition of all $h(q_j)$s results in $H(Q_i)$, which eventually leads to $H(Q)$.

Hence, our framework reduces the application workload in two ways. First, the query requires to access only eight corners, regardless of the query size. If the indexing results fit in memory, then it limits the memory access. If the indexing results for each data partition has been stored in compressed files, then it limits the number of disk accesses. Second, the retrieval of each corner distribution involves addition of a small number of sub-ranges, and then adding/subtracting corner histograms. Since the number of sub-ranges at any location x is at most $\log x$, the computation required is less compared to raw data scan for large queries.

Also depending on their sizes, sub-ranges contribute differently to the resulting integral histogram. For example, the sub-range $(65,65)$ of size 1 contributes much less than the sub-range $(1,64)$ to the integral histogram of 65. The user can exploit this principle to optimize query performance. We allow the user to select a *truncation threshold* T. All sub-ranges having number of element less than are discarded T while reconstructing an integral histogram. $T=0$ leads to exact reconstruction. $T>0$ results in approximate results. The smaller sub-ranges contribute less, but they are large in number, so discarding the smaller ones reduces the memory access and hence response time.

4 Applications

The proposed technique benefits any visualization application which implicitly runs range distribution queries on the data. Applications mainly run three types of distribution queries: First, while interactively exploring the spatial domain, the user places queries in *random* locations and sizes in no particular order. Second, a number of applications partition the data into blocks and require the distributions of each block as a substitute of downsampled data. Third, Some applications compute and analyze distributions at every voxel based on a small neighborhood around it. Our method seamlessly integrates with any visualization application which falls in one or more of the above three categories.

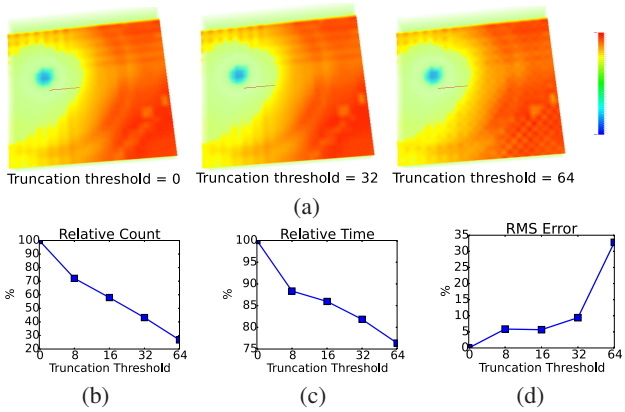


Figure 8. Local statistical analysis at different levels of detail. (a) Accurate and approximate reconstruction of block-wise mean field of Isabel pressure, (b) relative amount of memory access (in terms of sub-ranges access), (c) relative computation time, and (d) accuracy (measured by RMS error) of local statistical analysis at different levels of approximation.

4.1 Analysis of Local Statistics

Local statistics such as mean, variance and information entropy [23] are computed from block level distributions, or from distributions computed based on a local neighborhood at each point. Computation of such queries across large domains is expensive, and needs multiple access to data if the user wants to interactively change the level of details - either the block size, or the neighborhood size. On the other hand, our method can quickly recompute the distributions and the desired metric if the level of detail is changed. Left of Figure 8a shows such a mean field computed using our method from 8^3 blocks of Isabel Pressure field.

The middle and right images of Figure 8a show how our method can also generate approximate yet fairly accurate statistic fields while using much less query workload in terms of memory and decoding time. This enables quick analysis of larger datasets. We have generated this mean field at different truncation levels T between 8 and 64 and compared the query performance against the exact $(T=0)$ query result. Figure 8a shows how the memory footprint, measured by the number of sub-ranges accessed during reconstruction, falls off with T. At $T=64$, the query is answered only using 25% of the sub-ranges. As expected, the distribution computation time goes down as well (Figure 8b). The I/O time (not shown) would go down as well if different sub-ranges are organized in different files. In addition to showing that the approximated field is visually similar to the accurate one even at $T=64$, we present in Figure 8c the average RMS error for each level of approximation, compared with respect to the accurate mean field.

4.2 Feature Detection with Fuzzy Isosurfaces

Blockwise distribution queries are used in many other applications. To give an example, Thompson et. al. [22] has shown that fuzzy isosurfaces computed from block-level distributions retain the main features present in the original data. When raw data is not available, their technique computes a *likelihood value* for each block for a given isovalue. However, the user often needs to visualize these likelihood values for different size blocks. This either requires repeated data access to compute distributions for blocks of varying sizes, or requires storing of pre-computed block distribution hierarchy for a range of sizes. Both are impractical for large data.

In this situation, our method can retrieve the block distributions for any level of detail on demand, without touching the raw data

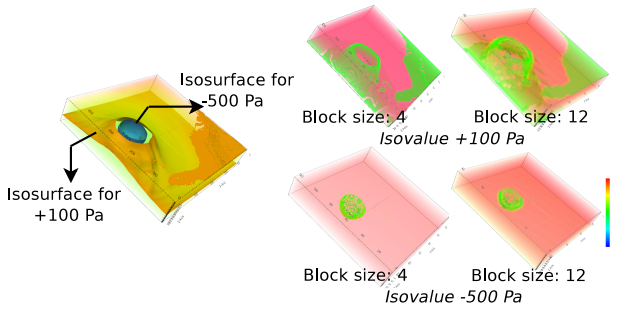


Figure 9. Block distribution-based fuzzy isosurface computation from Isabel Pressure field. **Left.** True isosurfaces. **Top Right.** Likelihood field for +100 Pa. **Bottom Right.** Likelihood fields for -500 Pa.

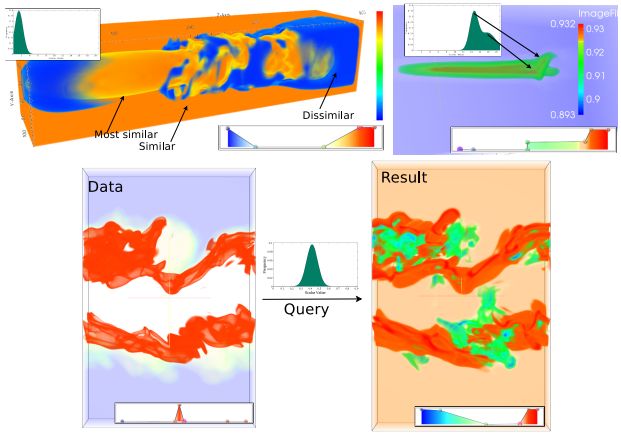


Figure 10. **Top.** Distribution-based search for different patterns on Solar Plume dataset (using a 9^3 local neighborhood). A commonly occurring distribution **(top left)**, and a sparsely present distribution **(top right)** used as template. **Bottom.** Distribution-based search for locations with stoichiometric mixture rate of 0.42 on time step 118 of combustion dataset (using a 7^3 local neighborhood).

and having to store a large number of distributions. We have shown an example using the pressure field of hurricane Isabel dataset. According to previous research [6], pressure values less than -350 Pa correspond to the hurricane eye. The top image in Figure 9 shows the true isosurface for isovalue -500 Pa, (blue surface), which surrounds the hurricane eye, and the isosurface for +100 Pa (orange surface), which spans a large area outside the eye. The bottom row contains the likelihood field for isovalues -500 Pa (left two) and +100 Pa (right two) computed from block sizes 4, and 12.

4.3 Distribution-based Similarity Search

The third application is an example of pointwise range distribution query. When the user does not know the exact isovalue to look for, it is more convenient to place the query in the form of a distribution, a Gaussian kernel centering at the best estimated isovalue or a Gaussian mixture for example. Also, some features are characterized by a particular type of distribution by the domain experts In such a context, a *fuzzy similarity score* proposed by Johnson and Huang [10] indicates the similarity between the user-input distribution and a local neighborhood based distribution at each voxel. The resulting similarity field has a score range from 0 to 1, 1 indicating the best match. However, whenever the user changes the neighborhood size to be used for search, this method has to re-compute the distributions at each voxel. This is why our method can enhance such a search by providing the ability to reconstruct the local distributions of different neighborhood sizes on the fly.

Table 1. Size of test datasets and the storage cost of corresponding IDVs (64 bins)

Dataset	Resolution	IDV Size (GB)
Plume	$126 \times 126 \times 512$	3.29
Isabel	$250 \times 250 \times 50$	1.49
Combustion	$240 \times 360 \times 60$	2.47

Figure 10 top row shows two examples of distribution based search on Solar Plume. The query for the top image (distribution shown in inset) is a more common one, so most voxels of the resulting field have higher similarity score (red or orange) and only a few regions are dissimilar to the query (shown in blue). On the contrary, the query distribution for the bottom image is a Gaussian mixture where both Gaussians are centered at less probable values. Hence, most of the voxels in the similarity field are 0 or close to 0 (shown in blue). We have modulated the opacity to highlight the two small regions corresponding to the two Gaussian peaks in the query. Figure 10 bottom shows a distribution query on mixture fraction variable of Combustion data. We have used a Gaussian distribution centered at 0.42 as the query. The high similarity zones in the search result (red, right image) correspond well to the flame extinction zones (value $\sim$0.42) in the raw data (red, left image).

5 Quantitative Analysis

In this section, we study the reduced storage cost and query workload achieved by our framework, and compare our technique with alternate approaches.

Datasets: Table 1 lists the sizes of the datasets used in our experiments. It also presents the size of the integral distribution volume to highlight the huge storage cost associated with it.

Assuming that loading the entire IDV to memory is not permissible for actual large datasets, we have partitioned these datasets into 1024 blocks and performed both the pre-processing and the query in parallel on blocks. Our primary focus is not on the parallelism. But we have worked with partitioned data to make our algorithm independent of total data size.

Alternate Methods: In terms of storing IDV, we have compared against direct compression of IDV using any off-the-shelf compression scheme. As a second alternative, we have run the proposed indexing algorithm directly on the IDV, bypassing the decomposition into power-of-two length sub-ranges, and compressed the indexing output. This alternative is presented to justify the importance of the decomposition step in our framework. Finally, we have results from our method which performs decomposition into sub-ranges, then indexing of sub-ranges and finally, compression.

The third stage – compression – works with any compression scheme. We have presented results using standard implementations of LZ (Lempel-Ziv) [24] and bzip2 algorithms. Our work does not recommend any particular compression algorithm though. Our objective is to transform IDV, which is originally not easy to compress, to an easily compressible form.

In terms of query performance, we have first compared against raw data access method, which is a simple load-and-filter approach. Given a range query, each data block that falls within the query is loaded from disk and scanned to compute the partial or full distribution. All these partial distributions are then combined to obtain the final query result. We have also compared against direct query of integral histogram method. In this case, we have retrieved the query distribution from the pre-computed and stored integral histograms of query corners. With no decoding involved, this one should be faster, albeit with an elevated storage cost.

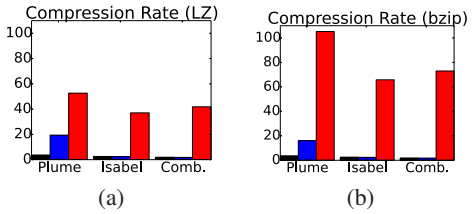


Figure 11. Compression rates of our method (right bars in each group) and two other methods: direct compression of IDV (left bars) and compression after direct indexing of IDV (middle bars).

5.1 Space Saving

The primary benefit of our method is that it enables the use of integral distributions with reduced storage cost. Each sub-range histogram is represented by an index to the corresponding template in a codebook, an integer and a boolean variable representing the transformation. The residuals of all sub-range histograms can be further compressed with conventional algorithms such as LZ or bzip2 since most of the bins in the residuals are zeros or small numbers.

It is noteworthy that our implementation divided the data into blocks so each block can be indexed and queried in parallel. For each block, the indices and the transformations for the sub-range histograms are stored in its own codebook. Only one set of templates is used for all blocks. The number of templates to be stored for Plume, Isabel and Combustion data are 403, 249 and 269, respectively, which are negligible compared to the total number of sub-range histograms. It should also be noted that for each block, we store the integral histograms of the boundary layers to its neighboring blocks so that the global integral histogram can be computed without recursively accessing the span histograms in other blocks.

Figure 11 presents the compression rates obtained by our method and compares it against two other techniques. The compression rates are measured with respect to the storage cost of the actual IDV. For each group, the leftmost bar indicates the space saving achieved when the IDV is directly compressed using a lossless compression technique such as LZ and bzip2. The middle bar in each group is the result of skipping the sub-range transformation and directly running our indexing algorithm. However, the results indicate that the original distributions from IDV do not lend themselves very well to an indexing algorithm. This justifies the necessity of the transformation into sub-ranges. Finally, we show that our method, combining transformation, indexing and compression, leads to very high compression rates (the rightmost bars in each group).

5.2 Query Response

According to our hypothesis, the performance of query in our method should not depend on query size, as we use integral histogram. We have tested this hypothesis by varying the query size from 16^3 to 80^3. For each query size, we have used a set of 2000 queries coming from different locations of the data. The performance numbers are based on an Intel(R) Core(TM) i7-2600 3.4GHz quad-core processor with 16GB memory. For all our experiments, each dataset is partitioned into 1024 blocks, and the query is computed on 8 processors in parallel. As the integral histograms can be large, our implementation loads the blocks needed for each query from disks on the fly.

Figure 12 depicts the performances of raw data access, direct query of integral histograms, and our approach. It shows that as the query lengths increase, the performance of our method and integral histograms are stable, while both the I/O and CPU time taken by raw data access method varies linearly with query size. Meanwhile, as expected, our approach is slower than directly querying integral histograms because our approach requires an additional step

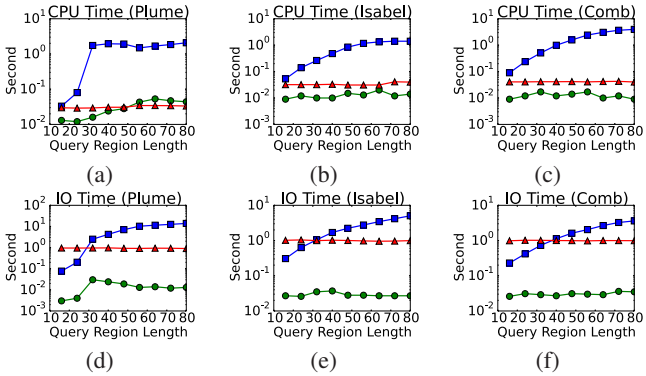


Figure 12. Performance analysis of query response for different query sizes and different datasets. The y axes in logarithmic scale. The tested algorithms are raw-access (□), integral histograms (◯), and our approach (△).

Table 2. Performance analysis of different stages of pre-processing (run on 8 processors in parallel). All times are in seconds.

Dataset	Template Creation	Sub-range Decomposition	LZ	BZ2
Plume	0.26	0.09	77.49	256.02
Isabel	0.07	0.04	188.59	194.77
Combustion	0.09	0.07	173.899	204.262

to decode and combine the sub-range distribution back to integral distributions. Nevertheless, as our approach requires much smaller storage than integral histograms (as already shown in Figure 11), it is suited for large datasets.

5.3 Performance Study

We have parallelized both the preprocessing and the query stages by partitioning the data into blocks and distributing the blocks across many processors. The preprocessing is easily adaptable to a parallel framework. In a parallel setting, each processor simultaneously computes its own list of templates based on the blocks assigned to it. A global list of templates is then created by accumulating the local lists. The global list is then distributed back to every processor. A global list is used for indexing because even if two blocks are spatially far apart, they may contain similar distributions. After template creation, each data block is decomposed into sub-ranges and then indexed in its own local co-ordinates space. To compensate for this local transformation, the integral distributions (in global coordinates space) for the corner point, three faces and three edges of the preceding blocks are also stored. Table 2 presents the running times of these stages and the compression time.

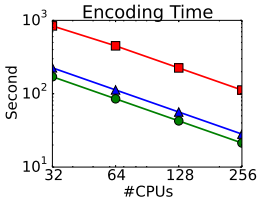


Figure 13. Performance analysis of indexing of power-of-two sub-range distributions for Plume (△), Isabel (◯), and Combustion (□). Each is partitioned into 1024 blocks.

The main indexing algorithm is computationally expensive since it compares each sub-range against each template. We have tested the scalability of this stage on Surveyor, an IBM BG/P supercomputer at Argonne National Laboratory. Surveyor contains 1024 quad-core nodes, 2TB memory and utilizes the General Parallel File System (GPFS). Figure 13 shows that the indexing performance scales well with number of processors for all three datasets

6 Discussion

In essence, we propose transformations to the raw data to make it more compressible. The user should choose the right compression technique depending on whether speed or storage is more critical in the query phase. For example, a method with lower compression rate and faster decompression time (such as LZ) may suit interactive visualization applications, while a slower algorithm producing better compression such as BZ2 is more useful for generating static fields and images from very large data. Figure 11 allows us to compare compression rates of the two techniques we experimented with. bzip2 provides slightly better compression than LZ at the cost of performance (Table 2). The user can increase the compression levels (1 used) to save more space at the cost of time.

We have presented query performance without the decompression time. This is because our IDV size is 64 times the data size (64 bins used), and the codebook size (about 1% of the IDV) is smaller than even the raw data. Hence, under the same available resources, either both raw data and codebook can be loaded uncompressed, or both need to be decompressed. With our method, only a few locations are to be decompressed and read from the codebook of each block. On the other hand, full data blocks, or significant fractions of them are to be decompressed to access raw data.

For time-varying datasets, we observed that the sub-range histograms coming from subsequent time steps can be indexed with high space-saving using templates from the first time step only. For 29 time steps of solar plume data, the space saving (computed individually for each time step) falls gradually, but stays well above 90%. This result indicates that the assumption of distributions being similar to each other holds across time steps as well.

7 Conclusion and Future Work

In this paper, we propose a technique to support arbitrary range distribution query on volumetric data. Our proposed technique allows answering such queries while keeping the query workload and the associated storage cost very low, regardless of data and query size. Any visualization algorithm which needs distributions from different regions benefits from the proposed technique.

The proposed technique opens up many future directions. For example, similarity-based indexing should apply to higher dimensional distributions computed from multi-variate datasets. In the 2D case, the sub-range histograms themselves become image patches which also contain redundancy and self-similarity. With suitable adaptation to the transformation techniques, our technique should be useful for 2D or higher dimensional histograms as well. Another possible direction is to generalize the method for query regions of arbitrary shape. The challenge lies in approximately decomposing arbitrary shapes into a minimal set of axis-aligned queries to utilize integral histogram based methods. Besides, we also plan to support such query processing on GPU.

Acknowledgments

This work was supported in part by NSF grant IIS-1017635, IIS-1065025, US Department of Energy DOE-SC0005036, Battelle Contract No. 137365, and Department of Energy SciDAC grant DE-FC02-06ER25779, program manager Lucy Nowell. We would like to thank the anonymous reviewers for their sincere feedback.

References

[1] C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *Vis '97: Proceedings of the IEEE Conference on Visualization*, pages 167–173, 1997.

[2] F. Buccafurri, D. Rosaci, L. Pontieri, and D. Sacca. Improving range query estimation on histograms. In *ICDE '02: Proceedings of the International Conference on Data Engineering*, pages 628 –638, 2002.

[3] H. Carr, B. Duffy, and B. Denby. On histograms and isosurface statistics. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1259–1266, 2006.

[4] V. Chandrasekhar, G. Takacs, D. M. Chen, S. S. Tsai, Y. Reznik, R. Grzeszczuk, and B. Girod. Compressed histogram of gradients: A low-bitrate descriptor. *International Journal of Computer Vision*, 96(3):384–399, 2012.

[5] F. C. Crow. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the Conference on Computer graphics and interactive techniques*, pages 207–212, 1984.

[6] L. Gosink, C. Garth, J. Anderson, E. Bethel, and K. Joy. An application of multivariate statistical analysis for query-driven visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(3):264–275, 2011.

[7] Y. Gu and C. Wang. Transgraph: Hierarchical exploration of transition relationships in time-varying volumetric data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2015 –2024, 2011.

[8] M. Hadwiger, R. Sicat, J. Beyer, J. Krüger, and T. Möller. Sparse pdf maps for non-linear multi-resolution image operations. *ACM Transactions on Graphics*, 31(6):133:1–133:12, 2012.

[9] A. Jacquin. Image coding based on a fractal theory of iterated contractive image transformations. *IEEE Transactions on Image Processing*, 1(1):18 –30, 1992.

[10] C. Johnson and J. Huang. Distribution-driven visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):734–746, 2009.

[11] G. Kindlmann and J. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *VolVis '98: Proceedings of the IEEE Symposium on Volume Visualization*, pages 79 –86, 1998.

[12] S. Liu, J. Levine, P.-T. Bremer, and V. Pascucci. Gaussian mixture model based volume visualization. In *LDAV '12: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization*, 2012.

[13] C. Lundstrom, P. Ljung, and A. Ynnerman. Local histograms for design of transfer functions in direct volume rendering. *IEEE Transactions on Visualization and Comp. Graphics*, 12(6):1570 –1579, 2006.

[14] S. Martin and H.-W. Shen. Transformations for volumetric range distribution queries. In *IEEE Pacific Visualization Symposium*, 2013.

[15] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. *SIGMOD Rec.*, 27(2):448–459, 1998.

[16] S. Nagaraj and V. Natarajan. Relation-aware isosurface extraction in multifield data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):182–191, 2011.

[17] M. Otto, T. Germer, and H. Theisel. Uncertain topology of 3d vector fields. In *PacificVis '11: Proceedings of the IEEE Pacific Visualization Symposium*, pages 67 –74, 2011.

[18] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB '97: Proceedings of the International Conference on Very Large Data Bases*, pages 486–495. Morgan Kaufmann Publishers Inc., 1997.

[19] F. Porikli. Integral histogram: a fast way to extract histograms in cartesian spaces. In *CVPR '05: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 829 –836, 2005.

[20] K. Potter, R. M. Kirby, D. B. Xiu, and C. R. Johnson. Interactive visualization of probability and cumulative density functions. *International Journal of Uncertainty Quantification*, 2(4):397–412, 2012.

[21] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *CVPR '07: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[22] D. Thompson, J. Levine, J. Bennett, P.-T. Bremer, A. Gyulassy, V. Pascucci, and P. Pebay. Analysis of large-scale scalar data using hixels. In *LDAV '11: Proceedings of the IEEE Symposium on Large Data Analysis and Visualization*, pages 23 –30, 2011.

[23] L. Xu, T.-Y. Lee, and H.-W. Shen. An information-theoretic framework for flow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1216 –1224, 2010.

[24] A. Ziv, J.; Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.